# Unique Sets Oriented Partitioning of Nested Loops with Non-uniform Dependences*

Jialin Ju and Vipin Chaudhary
Parallel and Distributed Computing Laboratory
Wayne State University, Detroit, MI
Phone: (313) 577-0605
Email: *vipin@eng.wayne.edu*

## ABSTRACT

*Although many methods exist for nested loop partitioning, most of them perform poorly when parallelizing loops with non-uniform dependences. This paper addresses the issue of parallelizing nested loops with non-uniform dependences. Our approach is based on convex hull theory, which has adequate information to handle non-uniform dependences. We introduce the concept of Complete Dependence Convex Hull, unique head and tail sets and abstract the dependence information into these sets. These sets form the basis of the iteration space partitions. The properties of the unique head and tail sets are derived using Convex Hull theory. Depending on the relative placement of these unique sets, the partitioning problem is grouped in to several cases. Several partitioning schemes are also suggested for implementing our technique. Preliminary implementation results of our scheme on the Cray J916 and comparison with other schemes show a dramatic improvement in performance.*

## INTRODUCTION

Loops with cross-iteration dependences can be roughly divided into two groups. The first group is loops with static regular dependences, which can be analyzed during compile time. Example 1 belongs to this group. The other group consists of loops with dynamic irregular dependences, which have indirect access patterns eg. loops used for edge-oriented representation of sparse matrices. These kind of loops cannot be parallelized at compile time, for lack of sufficient information.

[1]This work was supported in part by NSF MIP-9309489, US Army Contract DAEA-32-93-D-004 and Ford Motor Company Grant #0000952185

**Example 1:**

```
do i = 1, 12
   do j = 1, 12
      A(2*i+3, j+1) = ...
      ... = A(2*j+i+1, i+j+3)
   enddo
enddo
```

Static regular loops can be further divided into two sub groups; the ones with uniform dependences and the other with non-uniform dependences. The dependences are uniform only when the pattern of dependence vectors is uniform ie. the dependence vectors can be expressed by some constants which are *distance vectors*. Similarly we call dependences non-uniform when the dependence vectors are in some irregular pattern which cannot be expressed with *distance vectors*. Figure 1 shows the non-uniform dependence pattern of Example 1, which has a non-uniform dependence, in the iteration space,

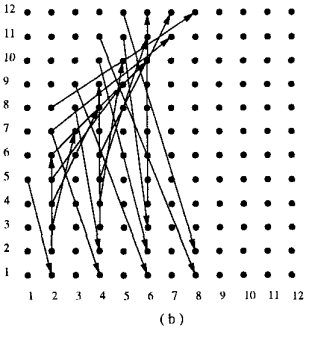


Figure 1: *Iteration space of Example 1*

In an empirical study, Shen *et al.* [1] observed that nearly 45% of two dimensional array references are coupled and most of these lead to non-uniform dependences. This paper focuses on parallelizing loops

with such dependences. Our approach is based on *Convex Hull* theory which has been proven [2] to have enough information to handle non-uniform dependences. Based on our Unique Set technique, we will divide the iteration space into several parallel regions, such that all the iterations in each region can be executed in parallel in most cases. In the worst case only the last region has to be run sequentially.

Research in parallelizing non-uniform nested loops has been limited. Tzen and Ni[3] proposed the Dependence Uniformization technique. This technique computes a set of basic dependence vectors using Dependence Slope theory and adds them to every iteration in the iteration space. This uniformization helps in applying existing partitioning and scheduling techniques, but imposes too many additional dependences to the iteration space. Our approach provides more accurate information about the iteration space and finds more parallelism. Two other techniques based on Convex Hull theory have been proposed recently. Zaafrani and Ito [4] proposed a three region approach which divides the iteration space into two parallel regions and one serial region. Punyamurtula and Chaudhary [5] use a Minimum Dependence Distance Technique to partition the iteration space into regular tiles. Our technique subsumes both the above techniques.

The rest of this paper is organized as follows. Section two describes our program model, reviews some fundamental concepts and introduces the concept of a Complete Dependence Convex Hull. Section three gives the definition of Unique Sets and methods to find them. Section four presents our Unique Set oriented partitioning technique. Section five confirms our claims with results comparing our technique with previously proposed techniques. Finally, we conclude in section six. Due to the space restrictions all the proofs of the theorems and corollaries have been omitted. Please refer to the technical report[6] for further details.

## PROGRAM MODEL AND DEPENDENCE REPRESENTATION

Studies [7, 1] show that most of the loops with complex array subscripts are two dimensional loops. In order to simplify explaining our techniques, our Program Model has a normalized, doubly nested loops with coupled subscripts (*i.e.*, subscripts are linear functions of loop indices). Both lower and upper bounds for indices should be known at compile time. Our general program model is:

do $i = L_1, U_1$
  do $j = L_2, U_2$
    $A(a_{11}i + b_{11}j + c_{11}, a_{12}i + b_{12}j + c_{12}) = \cdots$
    $\cdots = A(a_{21}i + b_{21}j + c_{21}, a_{22}i + b_{22}j + c_{22})$
  enddo
enddo

The most common method to compute data dependences involves solving a set of linear Diophantine equations with a set of constraints formed by the iteration boundaries. Given the program model above, we want to find a set of integer solutions $(i_1, j_1, i_2, j_2)$ that satisfy the system of Diophantine equations (1) and the system of linear inequalities (2) .

$$\begin{aligned} a_{11}i_1 + b_{11}j_1 + c_{11} &= a_{21}i_2 + b_{21}j_2 + c_{21} \\ a_{12}i_1 + b_{12}j_1 + c_{12} &= a_{22}i_2 + b_{22}j_2 + c_{22} \end{aligned} \quad (1)$$

$$\begin{cases} L_1 \le i_1 \le U_1 \\ L_2 \le j_1 \le U_2 \\ L_1 \le i_2 \le U_1 \\ L_2 \le j_2 \le U_2 \end{cases} \quad (2)$$

The Dependence Convex Hull(DCH) is a convex polyhedron and is a subspace of the solution space. Please refer to [3] for the definition.

There are two approaches to solving the system of Diophantine equations in (1). One way is to set $i_1$ to $x_1$ and $j_1$ to $y_1$ and solve for $i_2$ and $j_2$.

$$\begin{cases} i_2 = \alpha_{11}x_1 + \beta_{11}y_1 + \gamma_{11} \\ j_2 = \alpha_{12}x_1 + \beta_{12}y_1 + \gamma_{12} \end{cases}$$

where $\alpha_{11} = (a_{11}b_{22} - a_{12}b_{21})/(a_{21}b_{22} - a_{22}b_{21})$, $\beta_{11} = (b_{11}b_{22} - b_{12}b_{21})/(a_{21}b_{22} - a_{22}b_{21})$, $\gamma_{11} = (b_{22}c_{11} + b_{21}c_{22} - b_{22}c_{21} - b_{21}c_{12})/(a_{21}b_{22} - a_{22}b_{21})$, $\alpha_{12} = (a_{21}a_{12} - a_{11}b_{22})/(a_{21}b_{22} - a_{22}b_{21})$, $\beta_{12} = (a_{21}b_{12} - a_{22}b_{11})/(a_{21}b_{22} - a_{22}b_{21})$, $\gamma_{12} = (a_{21}c_{12} + a_{22}c_{21} - a_{21}c_{22} - a_{22}c_{11})/(a_{21}b_{22} - a_{22}b_{21})$

The solution space **S** is the set of points $(x, y)$ satisfying the equations given above. The set of inequalities can be written as

$$\begin{cases} L_1 \le & x_1 & \le U_1 \\ L_2 \le & y_1 & \le U_2 \\ L_1 \le & \alpha_{11}x_1 + \beta_{11}y_1 + \gamma_{11} & \le U_1 \\ L_2 \le & \alpha_{12}x_1 + \beta_{12}y_1 + \gamma_{12} & \le U_2 \end{cases} \quad (3)$$

where (3) defines a DCH denoted by **DCH1**.

Another approach is to set $i_2$ to $x_2$ and $j_2$ to $y_2$ and solve for $i_1$ and $j_1$.

$$\begin{cases} i_1 = \alpha_{21}x_2 + \beta_{21}y_2 + \gamma_{21} \\ j_1 = \alpha_{22}x_2 + \beta_{22}y_2 + \gamma_{22} \end{cases}$$

where $\alpha_{21} = (a_{21}b_{12} - a_{22}b_{11})/(a_{11}b_{12} - a_{12}b_{11})$, $\beta_{21} = (b_{12}b_{21} - b_{11}b_{22})/(a_{11}b_{12} - a_{12}b_{11})$, $\gamma_{21} =$

$(b_{12}c_{21} + b_{11}c_{12} - b_{12}c_{11} - b_{11}c_{22})/(a_{11}b_{12} - a_{12}b_{11})$, $\alpha_{22} = (a_{11}a_{22} - a_{12}b_{21})/(a_{11}b_{12} - a_{12}b_{11})$, $\beta_{22} = (a_{11}b_{22} - a_{12}b_{21})/(a_{11}b_{12} - a_{12}b_{11})$, $\gamma_{22} = (a_{11}c_{22} + a_{12}c_{11} - a_{11}c_{12} - a_{12}c_{21})/(a_{11}b_{12} - a_{12}b_{11})$

The solution space **S** is the set of points $(x, y)$ satisfying the solution given above. In this case the set of inequalities can be written as

$$\begin{cases} L_1 \leq & \alpha_{21}x_2 + \beta_{21}y_2 + \gamma_{21} & \leq U_1 \\ L_2 \leq & \alpha_{22}x_2 + \beta_{22}y_2 + \gamma_{22} & \leq U_2 \\ L_1 \leq & x_2 & \leq U_1 \\ L_2 \leq & y_2 & \leq U_2 \end{cases} \quad (4)$$

where (4) defines another DCH, denoted by **DCH2**.

We introduce a new term *Complete DCH* to represent the union of DCH1 and DCH2 (which were constructed by (3) and (4)) and we shall demonstrate that the Complete DCH contains all the information we need to parallelize the loop.

**Definition 1 (Complete DCH (CDCH))** *The Complete DCH is the union of two closed sets of integer points in the iteration space, which satisfy (3) or (4).*

We use an arrow to represent a dependence in the iteration space. We call the arrow's head *the dependence head* and the arrow's tail *the dependence tail.* Figure 2 shows the CDCH of Example 1.

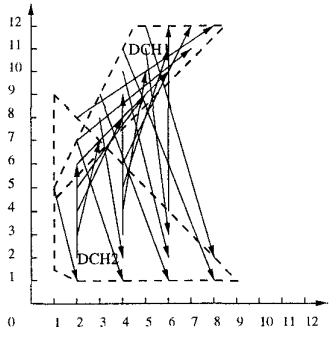


Figure 2: *CDCH of Example 1*

**Theorem 1** *All the dependence heads and tails lie within the CDCH. The head and tail of any particular dependence lie separately in the two DCHs of the CDCH.*

If iteration $(i_2, j_2)$ is dependent on iteration $(i_1, j_1)$, then we have a dependence vector D(x, y) with $d_i(x, y) = i_2 - i_1$, $d_j(x, y) = j_2 - j_1$. So, for DCH1, we have

$$\begin{aligned} d_i(x_1, y_1) &= (\alpha_{11} - 1)x_1 + \beta_{11}y_1 + \gamma_{11} \\ d_j(x_1, y_1) &= \alpha_{12}x_1 + (\beta_{12} - 1)y_1 + \gamma_{12} \end{aligned} \quad (5)$$

For DCH2, we have

$$\begin{aligned} d_i(x_2, y_2) &== (1 - \alpha_{21})x_2 - \beta_{21}y_2 - \gamma_{21} \\ d_j(x_2, y_2) &= -\alpha_{22}x_2 + (1 - \beta_{22})y_2 - \gamma_{22} \end{aligned} \quad (6)$$

Clearly, if we have a solution $(x_1, y_1)$ in DCH1, we must have a solution $(x_2, y_2)$ in DCH2, because they are derived from the same set of linear Diophantine equations (1).

## UNIQUE SETS IN THE ITERATION SPACE

As we have shown, all dependences lie within the CDCH. In other words, the iterations lying outside the CDCH are independent and can be executed in parallel. Hence we only have to worry about the iterations inside CDCH.

### UNIQUE HEAD AND UNIQUE TAIL SETS

DCH1 and DCH2 are our primitive sets. For a particular set it is possible that it contains both, dependence heads and tails.

**Definition 2 (Unique Head(Tail) Set)** *Unique head(tail) set is a set of integer points in the iteration space that satisfies the following conditions:*

1. *it is subset of one of the DCHs (or is the DCH itself).*
2. *it contains all the dependence arrow's heads(tails), but does not contain any other dependence arrow's tails(heads).*

Obviously the DCHs in Figure 2 are not the unique sets we are trying to find, because each DCH contains all the dependence heads of one kind and at the same time contains all the dependence tails of the other kind. Therefore, these DCHs must be further partitioned into smaller unique sets.

### FINDING UNIQUE HEAD AND UNIQUE TAIL SETS

We first examine the properties of DCH1 and DCH2.

**Theorem 2** *DCH1 contains all flow dependence tails and all anti dependence heads (if they exist) and DCH2 contains all anti dependence tails and all flow dependence heads (if they exist).*

The above theorem tells us that DCH1 and DCH2 are not unique head or unique tail sets. If there exist only flow or anti dependence, DCH1 either contains all the flow dependence tails or anti dependence heads, and DCH2 either contains all the flow dependence heads or anti dependence tails. Under these conditions, both DCH1 and DCH2 are unique sets. The following theorem states the condition for DCH1 and DCH2 to be unique sets.

**Theorem 3** *If $d_i(x,y) = 0$ does not pass through any DCH, then there is only one kind of dependence, either flow or anti dependence, and the DCH itself is the Unique Head set or the Unique Tail set.*

DCH1 and DCH2 are constructed from the same system of linear Diophantine equations and inequalities. The following two theorems highlight their common attributes.

**Theorem 4** *If $d_i(x_1,y_1) = 0$ does not pass through DCH1, then $d_i(x_2,y_2) = 0$ does not pass through DCH2.*

**Corollary 1** *When $d_i(x_1,y_1) = 0$ does not pass through DCH1,*

1. *if DCH1 is on the side of $d_i(x_1,y_1) > 0$, then*
   (a) *DCH1 is the flow dependence Unique Tail set, and*
   (b) *DCH2 is the flow dependence Unique Head set.*
2. *if DCH1 is on the side of $d_i(x_1,y_1) < 0$, then*
   (a) *DCH1 is the anti dependence Unique Head set, and*
   (b) *DCH2 is the anti dependence Unique Tail set.*

**Corollary 2** *When $d_i(x_1,y_1) = 0$ does not pass through DCH1,*

1. *if DCH1 is on the side of $d_i(x_1,y_1) > 0$, then DCH2 is on the side of $d_i(x_2,y_2) > 0$.*
2. *if DCH1 is on the side of $d_i(x_1,y_1) < 0$, then DCH2 is on the side of $d_i(x_2,y_2) < 0$.*

We have now established that if $d_i(x_1,y_1) = 0$ does not pass through DCH1, then both DCH1 and DCH2 are Unique Sets.

When $d_i(x,y) = 0$ passes through the CDCH, a DCH might contain both dependence heads and tails (even if DCH1 and DCH2 do not overlap). This makes it harder to find the unique head and tail sets. The next theorem looks at some common attributes when $d_i(x,y) = 0$ passes through the CDCH.

**Theorem 5** *If $d_i(x_1,y_1) = 0$ passes through DCH1, then $d_i(x_2,y_2) = 0$ must pass through DCH2.*

Using the above theorem we can now deal with the case where a DCH contains all the dependence tails of one kind and all the dependence heads of another kind.

**Theorem 6** *If $d_i(x,y) = 0$ passes through a DCH, then it will divide that DCH into a unique tail set and a unique head set. Furthermore, $d_j(x,y) = 0$ decides on the inclusion of $d_i(x,y) = 0$ in one of the sets.*

Note that if $d_j(x_1,y_1) > 0$, then the line segment corresponding to $d_i(x_1,y_1) = 0$ belongs to the flow dependence Unique Tail set and if $d_j(x_1,y_1) < 0$, then the line segment corresponding to $d_i(x_1,y_1) = 0$ belongs to the anti dependence Unique Head set. The iteration corresponding to the intersection of $d_i(x_1,y_1) = 0$ and $d_j(x_1,y_1) = 0$ has no cross-iteration dependence. If the intersection point of $d_i(x_1,y_1) = 0$ and $d_j(x_1,y_1) = 0$ lies in DCH1, then one segment of the line $d_i(x_1,y_1) = 0$ inside DCH1 is a subset of the flow dependence unique tail set and the other segment of the line $d_i(x_1,y_1) = 0$ inside DCH1 is a subset of the anti dependence unique head set.

For DCH2, we have similar results as above.

**Corollary 3** *When $d_i(x_1,y_1) = 0$ passes through DCH1, then*

1. *DCH1 is the union of the flow dependence Unique Tail set and the anti dependence Unique Head set, and*
2. *DCH2 is the union of the flow dependence Unique Head set and the anti dependence Unique Tail set.*

Figure 3 illustrates the application of our results to example 1. Clearly $d_i(x_1,y_1) = 0$ divides DCH1 into two parts. The area on the left side of $d_i(x_1,y_1) = 0$ is the Unique Tail set for flow dependence and the area on the right side of $d_i(x_1,y_1) = 0$ is the Unique Head set for anti dependence. $d_i(x_2,y_2) = 0$ divides DCH2 into two parts too. The area below $d_i(x_2,y_2) = 0$ is the Unique Head set for flow dependence and the area above $d_i(x_2,y_2) = 0$ is the Unique Tail set for anti dependence.

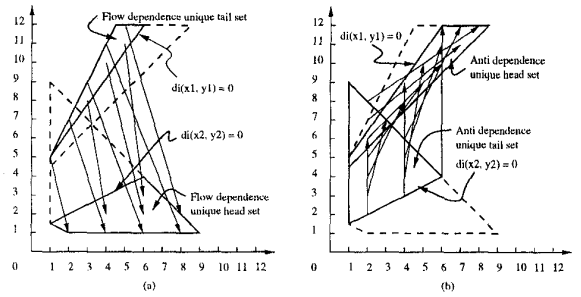


Figure 3: *Unique Head sets and Unique Tail sets of (a) Flow dependence, (b) Anti dependence*

## UNIQUE SETS ORIENTED PARTITIONING

Based on the Unique Head and Tail Sets that we identify, there exist various combinations of overlaps (and/or disjointness) of these Unique Head and Tail sets. We categorize these combinations as various cases starting from simpler cases and leading up to the more complicated ones. Because of lack of space we will not discuss all combinations. However, these combinations are not too many and can be generated systematically.

**Case 1** *There is only one kind of dependence and DCH1 does not overlap with DCH2.*

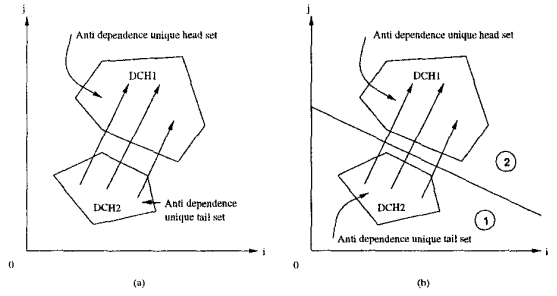


Figure 4: *One kind of dependence and DCH1 does not overlap with DCH2*

Figure 4(a) illustrates this case. Any line drawn between DCH1 and DCH2 divides the iteration space into two areas. The iterations within each area can be executed concurrently. However, the area containing DCH2 needs to execute before the area containing DCH1 (as shown by the partitioning in Figure 4(b)). The execution order is given by $1 \rightarrow 2$.

**Case 2** *There is only one kind of dependence and DCH1 overlaps with DCH2.*

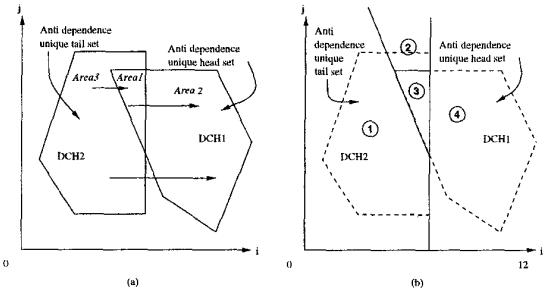


Figure 5: *One kind of dependence and DCH1 overlaps with DCH2*

Figure 5(a) illustrates this case. DCH1 and DCH2 overlap to produce three distinct areas denoted by $Area1$, $Area2$ and $Area3$, respectively. $Area2$ and $Area3$ are either Unique Tail or Unique Head sets and thus iterations within each set can execute concurrently. $Area1$ contains both tail and heads of dependences. One can apply the *Minimum Dependence Distance Tiling* technique proposed by Punyamurtula and Chaudhary [5] to $Area1$. Depending on the type of dependence there are two distinct execution orders possible. If DCH1 is a Unique Tail set, then the execution order is $Area2 \rightarrow Area1 \rightarrow Area3$. Otherwise, the execution order is $Area3 \rightarrow Area1 \rightarrow Area2$. Figure 5(b) shows one possible partitioning.

**Case 3** *There are two kinds of dependences and DCH1 does not overlap with DCH2.*

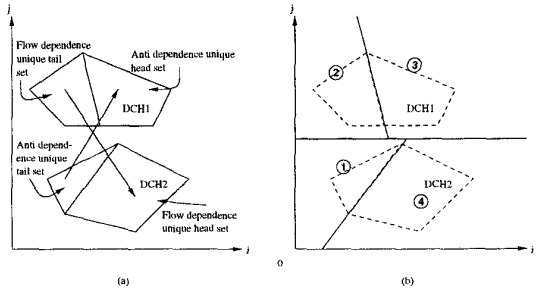


Figure 6: *Two kinds of dependences and DCH1 does not overlap with DCH2*

Figure 6(a) illustrates this case. Since DCH1 and DCH2 are disjoint we can partition the iteration space into two with DCH1 and DCH2 belonging to distinct partitions. From theorem 6 we know that $d_i(x, y) = 0$ will divide the DCHs into Unique Tail and Unique Head sets. We next partition the area with DCH1 by the line $d_i(x_1, y_1) = 0$, and the area with DCH2 by

the line $d_i(x_2, y_2) = 0$. So, we have four partitions, each of which is totally parallelizable. Figure 6(b) gives one possible partition with execution order as $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$. Note that the Unique Head sets must execute after the Unique Tail sets.

**Case 4** *There are two kinds of dependences and DCH1 overlaps with DCH2, and there is at least one isolated unique set.*

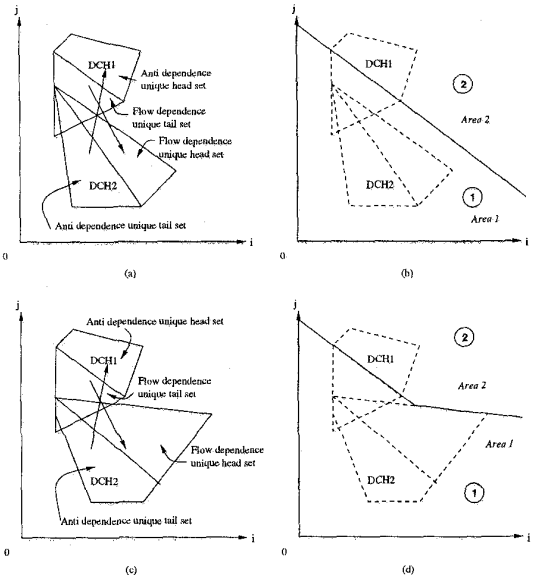


Figure 7: *Two kinds of dependences and one isolated unique set*

Figures 7(a) and 7(c) illustrate this case. What we want to do is to separate this isolated unique set from the others. The line $d_i(x, y) = 0$ is the best candidate to do this. If $d_i(x, y) = 0$ does not intersect with any other unique set or another DCH, then it will divide the iteration space into two parts as shown Figure 7(b). If $d_i(x, y) = 0$ does intersect with other unique sets or another DCH, we can add one edge of the other DCH as our boundary to partition the iteration space into two as shown in Figure 7(d). Let us denote the partition containing the isolated unique set by *Area*2. The other partition is denoted by *Area*1. If *Area*2 contains a unique tail set, then *Area*2 must execute before *Area*1, otherwise *Area*2 must execute after *Area*1. The next step is to partition *Area*1. Since *Area*1 has only one kind of dependence (as long as we maintain the execution order defined above) and DCH1 overlaps with DCH2, it falls under the category of case 2 and can be further partitioned as earlier.

**Case 5** *There are two kinds of dependence and all unique sets are overlapping with each other.*

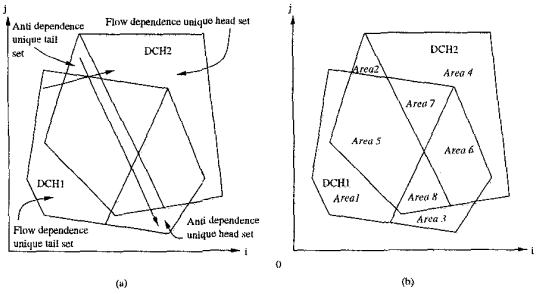


Figure 8: *Two kinds of dependence and all unique sets overlapped each other*

Figure 8(a) illustrates this case. The CDCH can be partitioned into at most eight parts as shown in Figure 8(b). *Area*1 contains only flow dependence tails. *Area*2 contains only anti dependence tails. *Area*3 contains only anti dependence heads. *Area*4 contains only flow dependence heads. *Area*5 contains flow dependence tails and anti dependence tails. *Area*6 contains flow dependence heads and anti dependence heads. *Area*7 contains flow dependence tails and flow dependence heads. *Area*8 contains anti dependence tails and anti dependence heads.

*Area*1, *Area*2, and *Area*5 can be combined together into a larger area, because they contain only the dependence tails. Let us denote this combined area by *AreaI*. In the same way, *Area*3, *Area*4, and *Area*6 can also be combined together, because they contain only the dependence heads. Let us denote this combined area by *AreaII*. *AreaI* and *AreaII* are fully parallelizable. The execution order becomes *AreaI* $\rightarrow$ *Area*7 $\rightarrow$ *Area*8 $\rightarrow$ *AreaII*. Since *Area*7 and *Area*8 contain both dependence heads and tails, we can apply *Minimum Dependence Distance Tiling* technique to parallelize this area.

## PRELIMINARY EXPERIMENTAL RESULTS

We executed Example 1 on a Cray J916 with 16 processors and 4 GBytes of RAM with different partitioning schemes. We analyzed the program using the *Autotasking Expert System* (atexpert), which is a tool developed by CRI for accurately measuring and graphically displaying tasking performance from a job run on an arbitrarily loaded CRI system. It can predict speedups on a dedicated system from data collected from a single run on a non-dedicated system.

We used *User-directed tasking* directives to construct our fully parallelizable area in the iteration space. We set the upper bounds of the loop to 1000.

The iteration space for Example1 is shown in Figure 9. In Figure 9(a), *Area*1 contains the flow dependence

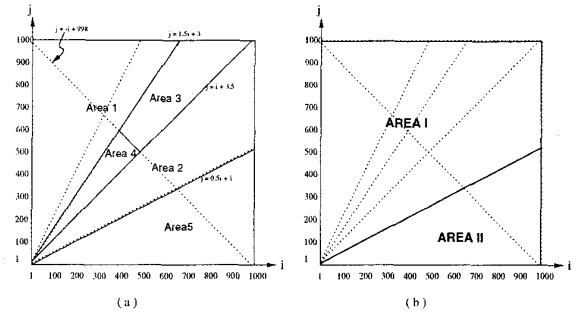


Figure 9: *Iteration space of Example 2*

tail set and part of the anti dependence tail set. *Area*2 contains only anti dependence tails. *Area*3 contains only anti dependence heads. *Area*4 contains both anti dependence heads and tails. *Area*5 contains the flow dependence head set. In figure 9(b), *AREA I* is the combination of *Area*1, *Area*2, *Area*3 and *Area*4, and *AREA II* is the same as *Area*5.

Obviously this example falls into the category of case 4. Based on unique sets, we can first isolate a parallel area which contains the flow dependence Unique Head set. That area is *Area*5 and it should execute last. The iteration space beyond this area, which is *AREA II*, falls into case 2. Now we need only consider one kind of dependence, anti dependence. At this point, we can either continue our partitioning according to the anti dependence Unique Head set and Tail set, or apply *minimum dependence distance tiling* technique to *AREA I*. We show these two approaches in the following schemes. In addition, we add the case where the last region of the first approach is run sequentially for comparison purposes (corresponding to scheme 2).

1. **Scheme 1:** executing in the order of *AREA I* → *AREA II*, where *AREA II* is fully parallel. In *AREA I* apply the *minimum dependence distance tiling* technique. Here minimum distance is 4 in the $j$ direction(*i.e.*, tiling size is 4 in the $j$ direction).

2. **Scheme 2:** executing in the order of *Area*1 → *Area*2 → *Area*3 → *Area*4 → *Area*5, where *Area*1, *Area*2, *Area*3 and *Area*5 execute in parallel and *Area*4 executes in serial order.

3. **Scheme 3:** executing in the order of *Area*1 → *Area*2 → *Area*3 → *Area*4 → *Area*5, where *Area*1, *Area*2, *Area*3 and *Area*5 execute fully in parallel. We apply the *minimum dependence distance tiling* technique to *Area*4(Here minimum distance is 4 too).

The results for the above schemes are shown in Figure 10, 11, and 12, respectively. The solid line is the

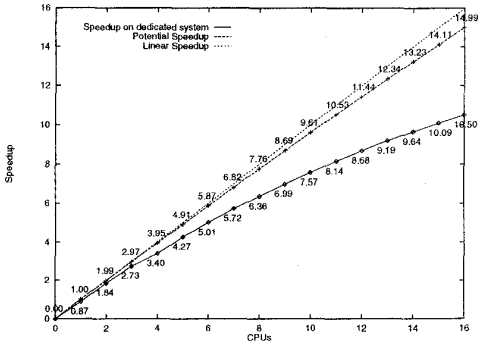


Figure 10: *Speedup for Scheme 1*

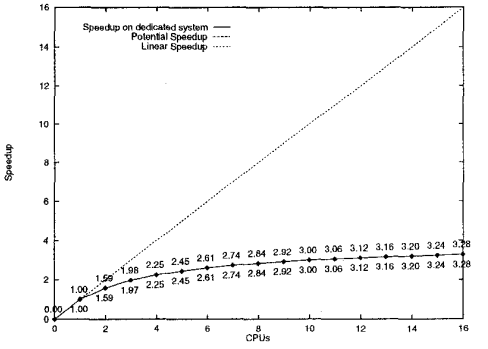


Figure 11: *Speedup for Scheme 2*

speedup on a dedicated system. The dashed line is the potential speedup which is the maximum speedup under ideal conditions for this program. The dotted line shows linear speedup, the boundary for all speedups.

Zaafrani and Ito's three-region region method[4] was also implemented and the result is shown in Figure 13. Clearly all three partitioning schemes we proposed show better results than the three-region method. The sequential region in our scheme is much smaller than in theirs.

Among our three schemes, Scheme 2 is the worst, showing 74.1% parallel and 25.9% serial parts. Scheme 1 comes in second, with 99.6% parallel and 0.4% serial

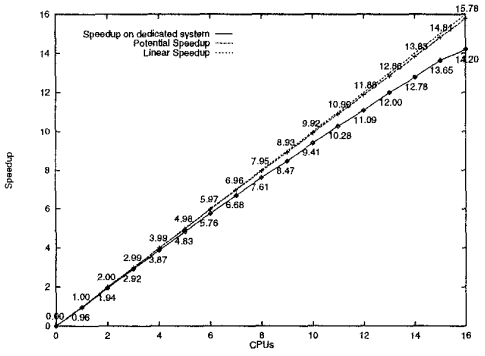


Figure 12: *Speedup for Scheme 3*

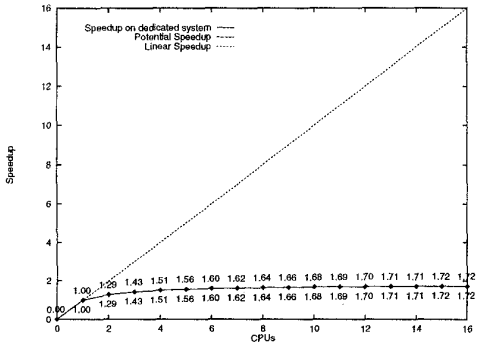


Figure 13: *Speedup for Zaafrani and Ito's three-region method*

parts. Scheme 3 is the best, it shows 99.9% parallel and 0.1% serial parts. The reason that Scheme 2 is worst is that it has a sequential region in the iteration space, which becomes a bottleneck. Scheme 3 is far better than Scheme 2, leaving nothing in the iteration space that would run sequentially. Scheme 1 uses tiling technique in *AREA I*, which divides *AREA I* into many small, fully parallel regions. These tiles would run sequentially. Scheme 3 shows almost linear speedup, since there are no sequential regions.

From our preliminary results, it appears that the iteration space should be partitioned into parallel regions based on unique sets and the non-parallelizable region should be partitioned using the minimum dependence distance tiling technique.

## CONCLUSION

In this paper, we systematically analyzed the characteristics of the dependences in the iteration space with the concepts of Complete Dependence Convex Hull, Unique Head sets and Unique Tail sets, which isolated the dependence information and showed the relationship among the dependences. We also proposed the Unique sets oriented partitioning of the iteration space. The suggested scheme was implemented on a Cray J916 and compared with the three-region technique. Preliminary results exhibit marked improvement in speedups.

## ACKNOWLEDGMENTS

We would like to thank Sumit Roy and Chengzhong Xu for their constructive comments on the contents of this paper.

## References

[1] Z. Shen, Z. Li, and P. C. Yew, "An empirical study on array subscripts and data dependencies," in *Proceedings of the International Conference on Parallel Processing*, pp. II–145 to II–152, 1989.

[2] Y. Q. Yang, C. Ancourt, and F. Irigoin, "Minimal data dependence abstractions for loop transformations: Extended version," *International Journal of Parallel Programming*, vol. 23, no. 4, pp. 359–388, 1995.

[3] T. H. Tzen and L. M. Ni, "Dependence uniformization: A loop parallelization tehnique," *IEEE transactions on Parallel and Distributed Systems*, vol. 4, pp. 547–558, May 1993.

[4] A. Zaafrani and M. Ito, "Parallel region execution of loops with irregular dependences," in *Proceedings of the International Conference on Parallel Processing*, pp. II–11 to II–19, 1994.

[5] S. Punyamurtula and V. Chaudhary, "Minimum dependence distance tiling of nested loops with non-uniform dependences," *Symp. on Parallel and Distributed Processing*, pp. 74–81, 1994.

[6] J. Ju and V. Chaudhary, "Unique sets oriented partitioning of nested loops with non-uniform dependences," *Technical Report PDCL 96-03-39*, Parallel and Distributed Computing Laboratory, Wayne State University, 1996.

[7] Z. Li, P. Yew, and C. Zhu, "An efficient data dependence analysis for parallelizing compilers," *IEEE transactions on Parallel and Distributed Systems*, pp. 26–34, Jan. 1990.